CSE 517, Winter 2013: Assignment 4

Adapted from Dan Klein's CS 288 Course at UC Berkeley.

Overview

In this assignment, you will implement three phrase-based decoders and test them with the provided harness. The data and support code are available on the course Dropbox in Catalyst.

What to Submit

Please submit two files:

- your writeup
  - as a PDF
  - with your name and username at the top
- a single archive (e.g. zip, tar.gz) containing
  - your source code
  - your compiled binary files

Do *not* submit the data.

Testing Decoders

The testing harness we will be using is `MtDecoderTester` (in the `edu.berkeley.nlp.assignments.assign2` package). To run it, first unzip the data archive to a local directory (`PATH`). Build the submission jar using `ant -f build_assign2.xml`. Then, try running

```
java -cp assign2.jar:assign2-submit.jar -server -mx2000m
   edu.berkeley.nlp.assignments.assign2.MtDecoderTester
   -path PATH -decoderType MONO_GREEDY
```

You will see the tester do some translation of 100 French sentences, which will take a few seconds, printing out translations along the way (note that printing of translations can be turned off with `--noprit`). The decoder is a very simple monotonic decoder which greedily translates foreign phrases one at a time. Its accuracy should be poor (BLEU score around 20), as should its model score (around 5216; see below for more about the model).

Implementing Decoders

Your task is to implement three decoders and test them. Each of your decoders will attempt to find the translation with maximum score under a linear model. This linear model consists of three features:

1. a trigram language model
2. a linear distortion model

3. a translation model which assigns scores to phrase pairs

Each of these features is weighted by some pretuned weights, so you should not need to worry about these weights (though you may experiment with them if you wish). The translation phrase pair table (with scores for each pair) and language model are provided, so you can focus on decoding.

You will need to implement three decoders of increasing complexity. We provide results of one specific implementation to guide you, without specifying exactly how it works. You are *not* required to get these exact numbers. However, each decoder should show improvement in both model score and BLEU score over the previous one. The only exception is the first one, which is expected to have lower performance when compared to the provided baseline.

To improve performance, you will adopt more complex models, but might also need to implement better beam and lattice management strategies. You are welcome to implement any approaches you want, including for example using different beams per subsets of foreign words or the lattice approach in the Collins notes, but should be careful to document what you did in the writeup.

You first two decoders will be monotonic, while the third will allow distortion. In a monotonic decoder, no distortion is allowed. Specifically, you must select phrases to translate from the foreign sentence in a left to right fashion, without skipping any foreign words at each step. When you add distortion, such skipping will be allowed.

Decoder 1: Monotonic (Without Language Model)

The first decoder should implement a monotonic search with no language model. You should return an instance of such a decoder from `MonotonicNoLmDecoderFactory`. Using our implementation, with beams of size 2000, we decode the test set in 16 seconds and achieve a model score of 5276 and BLEU score of 17.2. Note that this decoder performs poorly even compared to the greedy decoder since no language model is used. Remember, given your choice of strategy for managing the beam(s) or lattice, you should not expect to match these numbers exactly. But, it would be surprising if your numbers are significantly lower, and it is definitely possible that you could do better!

Decoder 2: Monotonic (With Language Model)

The second decoder should be constructed in `MonotonicWithLmDecoderFactory`. It should implement monotonic search with an integrated trigram language model. Using our implementation, with beams of size 2000, we decode the test set in 2 minutes and achieve a model score of 3781 and BLEU score of 26.5.

Decoder 3: Non-Monotonic (With Language Model and Distortion)

The third decoder should be constructed in `DistortingWithLmDecoderFactory`. It should implement a search strategy that permits limited distortion, as discussed in class. The distortion score can be retrieved from `DistortionModel` class, as can the distortion limit. Using our implementation, using

beams of size 2000, we decode the test set in about 20 minutes and achieve a model score of 3529 and BLEU score of 27.5.

Coding Tips

In general, expect the decoders to be slower as they become more complex. Pay attention to your implementation, or you will end up with a very slow development process. One data structure that is key to this kind of work is `PriorityQueue`. Unfortunately, The implementation provided by Java is relatively slow. However, we provide faster implementations. Look for the interface `edu.berkeley.nlp.util.PriorityQueue` and the classes implementing it. We strongly recommend using it.

Evaluation Metrics

The task of an MT decoder is to take a foreign sentence and find the highest-scoring English translation according to a particular model. If the model is a good one, then translation accuracy (BLEU) will correlate with model score, though this will not always be the case. As such, we will primarily evaluate your decoder on what model score it achieves on the training data, though we will measure BLEU as well. Also, with all decoders, there is a speed vs. accuracy tradeoff: you can always achieve a higher model score with a slow decoder, or decode very quickly if you do not care about making large numbers of search errors and achieving a low model score.

The test harness will return BLEU scores, a model scores, and decoding time.

To generate results you will need to run the following commands:

```
java -cp mt-lib.jar:mt-submit.jar -server -mx2000m
  edu.berkeley.nlp.assignments.decoding.MtDecoderTester
  -path PATH -decoderType MONO_NOLM
```

```
java -cp mt-lib.jar:mt-submit.jar server -mx2000m
  edu.berkeley.nlp.assignments.decoding.MtDecoderTester
  -path PATH -decoderType MONO_LM
```

```
java -cp mt-lib.jar:mt-submit.jar -server -mx2000m
  edu.berkeley.nlp.assignments.decoding.MtDecoderTester
  -path PATH -decoderType DIST_LM
```

Before you submit your code, please verify that these commands execute properly.

Writeup

Include the following things in your writeup. Answer questions with at most three sentences (unless otherwise indicated).

Part 1: Results

1. Plot decoder type (1-3) versus BLEU score.

2. Plot decoder type versus decode time.
3. Plot decoder type versus model score.
4. For decoder 3 (Non-Monotonic with language model and distortion), plot the beam size (10, 100, 1000) versus BLEU score.
5. For decoder 3, plot the beam size (10, 100, 1000) versus runtime.

Part 2: Decoding Model

Answer the following questions about the decoding model used by decoder 3.

1. What are the states in your decoding model (e.g. full state, compact state, or something else)?
2. What data structure do you use to represent a state?
3. Given a state, how do you compute its successor states (i.e. how do you implement the `next` function on the *Decoder Pseudocode* lecture slide)?
4. Yes or No + one sentence justification: Will there always be a complete translation of the input sentence?

Part 3: Decoding Algorithm

Answer the following questions about your implementation of decoder 3.

1. What data structures do you use to manage your beam (e.g. a single priority queue, multiple priority queues, or...)?
2. What formula do you use to score partial translations?
3. Yes or No + one sentence justification: Is your algorithm *complete* (it will always find a translation if one exists)?
4. Yes or No + one sentence justification: Is your algorithm *optimal* (it will always find the best translation if it exists)?

Part 4: Output Analysis

1. Find a common error type in your system and describe it.
2. Describe how you would fix this error (e.g. change the state formulation, change the decoding algorithm, change the scoring function, change the phrase table, etc.).